HW

NUMBER 1)

int main()

{

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 30

\*(ptr + 1) = 20; // set arr[1] to 20 //CHANGED

ptr += 2; //;pointer points to 20

ptr[0] = 30; // set arr[2] to 10 //CHANGED

while (ptr >= arr)

{

//ptr--;

cout << \*ptr << endl; // print values

ptr--; //MOVED POINTER HERE

}

}

1. The function won’t set the pToMax parameter to point to the maximum item in the array. This is because the parameter only changes a copy of the pointer, but does not actually change the variable within the findMax function.

void findMax(int arr[], int n, int\* &pToMax) //CHANGE AT REFERENCE HERE

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr;

for (int i = 1; i < n; i++)

{

if (arr[i] > \*pToMax)

pToMax = arr + i;

}

}

int main()

{

int nums[4] = { 5, 3, 15, 6 };

int\* ptr;

findMax(nums, 4, ptr);

cout << "The maximum is at address " << ptr << endl;

cout << "It's at position " << ptr - nums << endl;

cout << "Its value is " << \*ptr << endl;

}

1. This function may not work because the pointer needs to be initialized to the address of a certain integer. Otherwise, the pointer is not pointing to the intended position and the function would not work.

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

//NEED TO POINT TO AN ARRAY

int num; //CHANGED

int\* ptr = &num; //CHANGED

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

1. The strequal function has a problem with element/pointer confusion. The program was meant to check whether elements in a string were equal, but was instead checking whether pointers in a string are equal. As a result, the while loop would never be false and will create an infinite loop. To fix this problem I instead had the program check for whether the elements POINTED to by the the pointers were equal or not.

// return true if two C strings are equal

bool strequal(const char str1[], const char str2[])

{

while (\*str1 != 0 && \*str2 != 0)

{

if (\*str1 != \*str2) // compare corresponding characters

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time?

}

int main()

{

char a[15] = "Noor";

char b[15] = "Noah";

if (strequal(a,b))

cout << "They're the same person!\n";

}

1. The problem is scoping. anArray doesn’t exist in the main function and the memory can be reused however. The space that anArray used can be used for anything else and as result will manipulate the memory that was not intended to be touched by the program.

NUMBER 2)

1. Declare a pointer variable named cat that can point to a variable of type double.

double\* cat;

1. Declare mouse to be a 5-element array of doubles.

double mouse[5];

1. Make the cat variable point to the last element of mouse.

cat = &mouse[4]; //cat is pointing to the address of 4 in mouse

cat = mouse + 4;

1. Make the double pointed to by cat equal to 42, using the \* operator.

    \*(cat) = 42;

1. Without using the cat pointer, and without using square brackets, set the fourth element (i.e., the one at position 3) of the mouse array to have the value 25.

    \*(mouse + 3) = 25;

1. Move the cat pointer back by three doubles.

    cat -= 3;

1. Using square brackets, but without using the name mouse, set the third element (i.e., the one at position 2) of the mouse array to have the value 17.

\*(cat+1) = 17;

1. Without using the \* operator, but using square brackets, set the double pointed to by cat to have the value 54.

    cat[0] = 54;

1. Using the \* operator in the initialization expression, declare a bool variable named b and initialize it to true if the double pointed to by cat is equal to the double immediately following the double pointed to by cat, and false otherwise.

    bool b = (\*cat == \*(cat +1));

1. Using the == operator in the initialization expression, declare a bool variable named d and initialize it to true if cat points to the double at the start of the mouse array, and false otherwise.

    bool d = (cat == (mouse));

NUMBER 3)



double mean(const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

for (int i = 0; i < numScores; i++)

{

tot += \*(ptr + i);

}

return tot/numScores;

}



// This function searches through str for the character chr.

// If the chr is found, it returns a pointer into str where

// the character was first found, otherwise nullptr (not found).

const char\* findTheChar(const char\* str, char chr)

{

for (int k = 0; \*(str+k) != 0; k++)

if (\*(str+k) == chr)

return str + k;

return nullptr;

}

const char\* findTheChar(const char\* str, char chr)

{

while (\*(str) !=0)

{

if(\*str == chr)

return str;

str++;

}

return nullptr;

}

NUMBER 4)

The program prints out the following numbers:

3

4

79

-1

9

22

19

To explain such, we must go through entire main function.

int\* ptr equals whatever is returned by the maxwell function. Since array is a pointer to the address of 0 and &array[2] is a pointer to the address of 2, it returns the position of the element with the greater value. This position is 0, so int\* prt is pointing to address of 0.

Since “\*prt = -1”, array[0] is initialized to -1.

“ptr += 2;” moves pointer by two spaces, so now it is pointing at array[2].

“ptr[1]” looks at one more position over in the array, array[3], initializing its element to to 9.

“\*(array+1)” looks at one position over from the current position arrayis pointing to, setting array[1] to 79.

For the first cout statement, “&array[5]” is at address of array[5] and subtracts “ptr” which has its current address at array[2]. The main function prints the result of 5-2, which is 3.

“swap1” only switches the addresses of array[0] and array[1], doing nothing to the elements pointed to by these elements. “swap2” switches array[0] and array[2], which now equal 4 and -1.

The for loop prints the modified elements of the array.

NUMBER 5)

void removeS (char\* msg) //msg already pointing to zero

{

for (; \*(msg) != '\0'; msg++) //running through message

{

if (\*msg == 's' || \*msg == 'S') //if element pointed to is s

{

char\* temp; //declared here for scope

for (temp = msg + 1; \*temp != '\0'; temp++) //running through leftover part of message to move s to the end

\*(temp - 1)= \*temp;

\*(temp - 1) = '\0'; //delete last element

msg--;

}

}

}